Dynamo

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

Motivation

- Reliability is one of the biggest challenges for Amazon.
- Amazon aims at 99.999% reliability (5 9s) (Less than 5 mins per year)
- Lack of reliability can translate into significant financial losses
- The infrastructure consists of thousands of servers
  - Servers and network components keep failing.
  - Customers need an always-on experience.

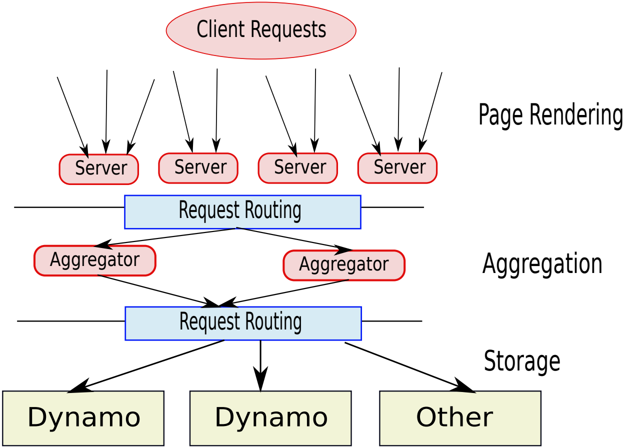
Dynamo

- Highly available key-value store.
- Serves a diverse set of applications
- Services – Best seller's list, shopping carts, customer preferences, sales rank, product catalog
- Served 3 million shopping checkouts in a single day
- Manages session state for thousands of concurrently active sessions
- Provides a simple key-value interface over the network

Assumptions and Requirements

- Query Model – Simple key-value access
- ACID properties – Provides only A , I , and D
- Latency requirements – 99.9% of all accesses satisfy the SLA
- SLA $\Rightarrow$ Service Level Agreement
  - Maximum Latency
  - Maximum client request rate

System Diagram



Key Principles of the Design

- Incremental Scalability : Should be able to scale one node at a time.
- Symmetry : Every node should have the same responsibility.
- Decentralization : Peer to peer system
- Heterogeneity : Needs to be able to exploit heterogeneous capabilities of servers.
- Eventual Consistency:
  1. A get operation returns a set of versions (need not contain latest value).
  2. A write ultimately succeeds.

Basic Operations

- get(key) Returns all the versions of an item (*context*).
- put(key,object, context) Add the object corresponding to the key in the database.

Partitioning

- Uses consistent hashing (similar to Chord) to distribute keys in a circular space.
- Each item is assigned to its successor.
- Uses the notion of virtual nodes for load balancing.
- A physical node is responsible for multiple virtual nodes.
- For fault tolerance, the key is assigned to N successors (preference list).
- One of these N successors, is the co-ordinator node.

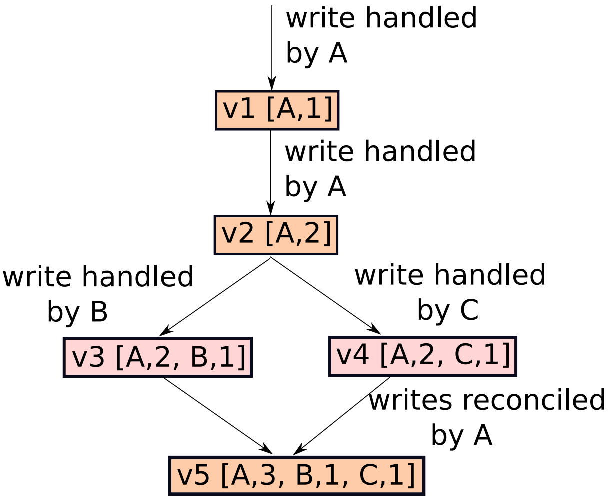
Data Versioning

- A put call might return before the update has propagated to all replicas.
- If there is a failure then some replicas might get the update after a very long time.
- Some applications such as “add to shopping cart” (write), need to always complete. (Prioritize Writes)
- Each new version of data is treated as a new and immutable version of data (recall Percolator).
- If there are failures and concurrent updates, then version branching may occur.
- Reconciliation needs to be performed among multiple updates
  - Can be done at the server side (generic logic)
  - Can be done at the client side (semantic merging)

Vector Clocks for Versioning

- A vector clock, contains an entry for each server in the preference list.
- When a server updates an object, it increments its vector clock.
- If there are concurrent modifications, then a get operation returns all versions.
- The put operation indicates the version.
- The put is considered a merge operation.
- Example ⇒

Vector Clock Example



Execution of get() and put()

- Send the request to any node that will forward it to the co-ordinator (like Pastry).
- Or, directly find the successor.
- The nodes ideally access the preference list (or top N healthy nodes)
- There is a read quorum of R nodes, and write quorum of W nodes
- $R + W > N$
- For a put request, the co-ordinator merges the versions, and broadcasts it to the quorum
- For a get request, the co-ordinator sends all the concurrent versions to the client

Sloppy Quorum

- Uses the first N healthy nodes (typically the preference list)
- If a node cannot deliver an update to node A, then it will send it to node D with a hint
- Once A recovers, D will transfer the object
- For added reliability the quorum spans across data centers

Synchronization across Replicas

- Nodes maintain Merkle trees $\Rightarrow$ The parent is the hash of the children.
- A Merkel tree contains the set of keys mapped to each virtual node per physical node.
- Nodes regularly exchange Merkle trees, through an anti-entropy based algorithm.
- Trees need to be often recalculated.

Maintaining Membership

- Dynamo maintains membership information through explicit join and leave requests.
- Ring membership changes are infrequent.
- Additionally a gossip based protocols propagates ring membership information across randomly chosen nodes.
- For 1-Hop routing nodes maintain large routing tables.
- All routing, membership, and placement propagates through an anti-entropy based gossip protocols.
- To prevent logical partitions, some nodes act as seeds , and synchronize information across peers.

Load Balancing and Failure Detection

- Failure detection is also done with gossip style protocols.
- Allocation and De-allocation happens in the same manner as Chord.

- Three different types of storage engines
  - In memory buffer with persistent backing store.
  - Berkely DB
  - MySQL DB
- Request co-ordination
  - Communication through Java NIO channels

Result: Read-Write Response Time

- In the peak season of December 2006.
- The average read time varied periodically (time period: 2 hours) between 12 to 18 ms.
- The average write time varied periodically (2 hours) between 21 to 30 ms.
- The 99 percentile values were roughly 10 times more.

Result: BDB vs Buffered Writes

- The 99.9^{th} percentile response time for buffered writes was between 40 and 60 ms.
- For direct BDB writes the fluctuations were much more (between 40 and 180 ms).

Discussion

- Reconciliation Methods
  - Business Logic Based Reconciliation : Shopping cart
  - Time stamp based Reconciliation (last write wins): Customer session management

Dynamo: Amazon's Highly Available Key-Value Store, Decandia et. al.